Crowdsourcing user interface adaptations for minimizing the bloat in enterprise applications

Crowdsourcing User Interface Adaptations for Minimizing the Bloat in Enterprise Applications

Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu
Computing Department, The Open University
Walton Hall, Milton Keynes, United Kingdom
{pierre.akiki, a.k.bandara, y.yu}@open.ac.uk

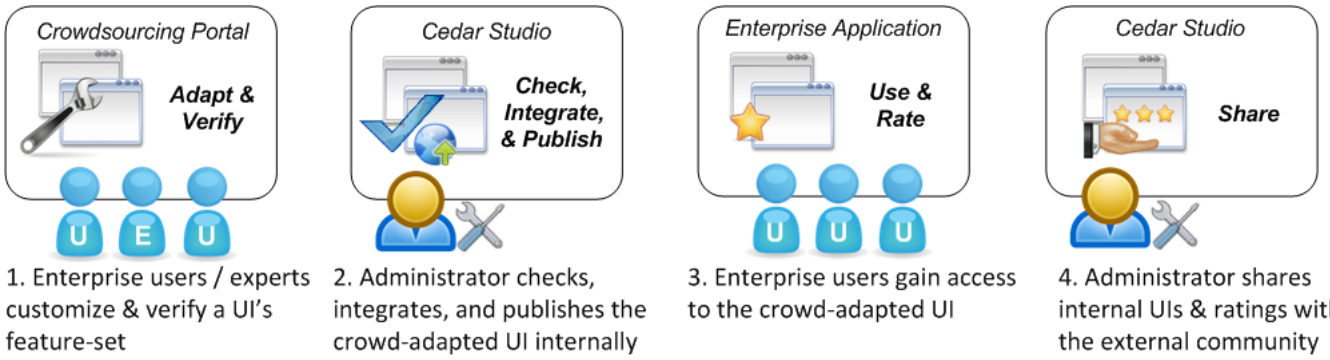

Figure 1. Our Process for Crowdsourcing Enterprise Application User Interface Adaptations

ABSTRACT
Bloated software systems encompass a large number of features resulting in an increase in visual complexity. Enterprise applications are a common example of such types of systems. Since many users only use a distinct subset of the available features, providing a mechanism to tailor user interfaces according to each user's needs helps in decreasing the bloat thereby reducing the visual complexity. Crowdsourcing can be a means for speeding up the adaptation process by engaging and leveraging the enterprise application communities. This paper presents a tool supported model-driven mechanism for crowdsourcing user interface adaptations. We evaluate our proposed mechanism and tool through a basic preliminary user study.

Author Keywords
Crowdsourcing; Adaptable user interfaces; Bloated UI; Enterprise applications; Model-driven engineering

ACM Classification Keywords
[Software Engineering]: D.2.11 Software Architectures - Domain-specific architectures; D.2.2 Design Tools and Techniques - User interfaces; [Information Interfaces and Presentation]: H.5.2 User Interfaces – User-centered design

General Terms
Design; Human Factors

Authors' Version
EICS'13, London, United Kingdom, *Forthcoming*

INTRODUCTION
The term "Bloat" [14] is used when referring to an excess of features in software applications leading to a diminished user experience [15]. Although enterprise applications (e.g., enterprise resource planning (ERP) systems, online retail stores, etc.) present the users with a large set of features, each user tends to use a different subset of them. This variation in user needs makes the concept of "Bloat" highly applicable to enterprise applications. Adapting a user interface's feature-set to the needs of individual users could greatly decrease its visual complexity [13].

The concept of crowdsourcing UI adaptations has been used by the gaming community to allow gamers to customize the user interface of a game level and share it with the rest of the community [9]. Leveraging this concept for enterprise applications could be beneficial when considering the large communities and commercial interests in these applications. We differentiate between the following two types of crowdsourcing for adapting enterprise application UIs:

- *Enterprise Crowdsourcing*: Allows internal enterprise staff members to adapt user interfaces
- *Community Crowdsourcing*: Leverages the external communities that use the same enterprise system

A combination of both types could be used for gaining the widest possible benefit from the crowd. An overview of our proposed process is illustrated in Figure 1 and will be further explained in the paper.

The model-driven approach to UI development [7] provides an interesting foundation for dealing with bloated UIs. We previously presented a mechanism called Role-Based User

Interface Simplification (RBUIS) [2] that provides the ability to minimize a UI's feature-set by assigning roles to tasks in task models hence achieving a multi-layer user interface design [19]. In RBUIS, roles are usually assigned by enterprise administrators using the *Cedar Studio* IDE and the end-users are given the ability to provide their feedback on the adaptations presented by the system. RBUIS is based on the CEDAR architecture [1], which promotes the use of interpreted runtime models for developing adaptive enterprise application user interfaces.

In this paper, we extend RBUIS by allowing end-users to perform the adaptation through a web-based feature-set editing tool, which can be made available online for enterprise community members. We should note that the technique presented in this paper complements RBUIS from the following perspectives: (1) End-users can adapt the feature-set using a simple tool without attaching the adaptations to user roles, afterwards administrators could attach the UI adapted by the crowd to one or more enterprise roles. This helps administrators in delegating some of the adaptation effort to the crowd. (2) The proposed technique is potentially helpful with non-role-based enterprise tools (e.g., word processors, spreadsheet managers, etc.) where the user could apply one of the crowd-adapted user interfaces based on a given context.

We consider the following criteria to be important in any approach targeting crowdsourcing enterprise application user interface adaptations:

- An web-based visual tool that allows various enterprise stakeholders (e.g., end-users, experts, etc.) to easily adapt the feature-set of user interfaces
- The ability to check whether the selected user interface features are consistent with the unselected ones according to the inherent dependencies
- A means for end-users to evaluate the usability of the crowd-adapted user interfaces
- Catering for enterprise privacy concerns by allowing administrators to control the UIs that are made available to the internal users and the ones that are shared with external communities

This paper makes the following contributions:

- A tool supported technique for crowdsourcing the adaptation of enterprise application UIs addressing the previously listed criteria
- An evaluation of the tool and technique through a basic online user study (with some limitations) that provided encouraging results in terms of the perceived usability and measured efficiency and effectiveness

RELATED WORK

This section briefly discusses existing works that target the minimization of the UI feature-set to fit various needs and the crowdsourcing of UI adaptations for engaging online software communities in the adaptation process.

Several research works target the adaptation of the UI feature-set such as: "*multi-layered UI*" [19], "*training wheels UI*" [8], "*two-interface design*" [13], "*MANTRA*" [5], etc. Yet, although crowdsourcing has been targeted by researchers for various purposes (e.g., performing expert work [10], human centered tasks such as image selection [4], etc.) few research works target crowdsourcing as a means for UI adaptation that engages and leverages the user communities behind software applications.

A primary advantage of our technique over existing works targeting the crowdsourcing of UI adaptations is the use of a model-driven approach that allows automatic model-checking to determine whether removing a feature affects other remaining features. This dependency is determined through the temporal operators in ConcurTaskTrees (CTT) [18] that are used to represent a UI's task model. Other researchers [3] have used a similar approach for checking CTTs for task dependency but our technique can be demonstrated by an algorithm (Appendix).

Adaptable Gimp [12] is presented as a socially adaptable alternative of the GNU image manipulation program Gimp. Adaptable Gimp allows the community to customize its UI by creating task-sets in a wiki. The work stresses on the importance of user feedback but no mechanism is provided for the users to evaluate the crowd-adapted UIs. Also, even though Gimp has a WIMP style UI, the adaptation focuses on a list of actions in the toolbar. Other research on UI adaption similarly focuses on drop-down menus [13]. Our aim is to be able to adapt any parts of the UI. Also, privacy concerns in terms of managing the adopted and shared UIs are not an issue in non-enterprise tools such as Gimp but our approach addresses these concerns by allowing administrators to control parts of the process.

Another approach [17] allows HTML based UIs to be adapted by users through a toolkit with a predefined set of adaptation operations. The changes are stored in a central repository as Cascading Style Sheets (CSS), which could be applied for other users with similar needs. This approach has several downsides: (1) It is technology dependent since the toolkit only works with HTML based web-pages whereas a model-driven approach provides technology independence. (2) Storing the customizations as CSS files makes operations such as model checking difficult.

One approach [16] attempts to involve the application's user community in the initial user interface design process by crowdsourcing the engineering of UIs. Although this approach has its advantages it does not tackle multi-context UI adaptation, which occurs at a post development stage.

The following section explains the steps, illustrated in Figure 1, of our process for crowdsourcing UI adaptations.

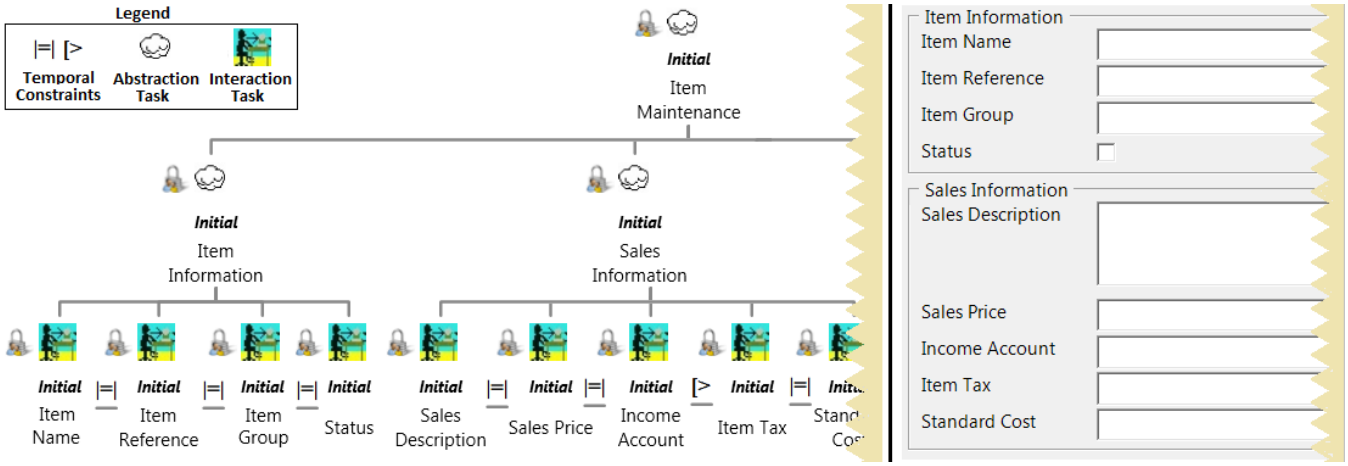

Figure 2. User Interface Task Model (Left) and Concrete UI Model (Right) created using Cedar Studio (Excerpt)

PROCESS OF CROWDSOURCING UI ADAPTATIONS

User interfaces can be represented in *Cedar Studio* on the levels of abstraction given by the CAMELEON reference framework [7]. The excerpt in Figure 2 shows two of these levels namely the task model (left) and the concrete UI (CUI) model (right) representing an "*Item Maintenance*" UI that is common in ERP systems. *Cedar Studio* stores the UI models in a relational database, which could serve as a repository for sharing these UIs among various enterprise stakeholders who can adapt the feature-set. The following subsections explain our process for crowdsourcing the adaptation of enterprise application UIs (Figure 1).

Step 1: Enterprise Stakeholders Adapt and Verify the UI

The adaptation process starts with enterprise stakeholders adapting a UI and executing an automatic verification to check whether the adaptation creates any conflicts. The stakeholders could be internal employees primarily wishing to adapt the UI for their personal use or external experts willing to contribute their experience to the community.

Any stakeholder can adapt UIs by using our web-based visual feature-set editing tool shown in Figure 3. Internal employees could connect the tool to their local enterprise database, whereas external experts could connect it to an online repository setup by the community for collaboration. The tool loads the task model as a tree structure (Figure 3 – Left), and dynamically renders the CUI using HTML (Figure 3 – Right). Stakeholders wishing to adapt the UI's feature-set could simply check/uncheck the selected task in the tree or click on the check/delete buttons next to each CUI element. Upon removing a parent task, the tool will automatically remove all of its subtasks. A description is given to the adapted UI to indicate the purpose of the adaptation (e.g., task, user's computer literacy, device, etc.).

The dependency between features could create conflicts when removing some while keeping others. For example, a conflict could happen if "Field A" was removed but "Field B" depends on it to calculate its value. The solution for such conflicts would be either removing or keeping both features. Upon completing the adaptation, it is possible to automatically verify the outcome. This verification relies on Algorithm 1 (Appendix) for checking if the removed tasks affect any other remaining tasks. Errors similar to the one shown in Figure 3 would be displayed with the option of reversing the action by re-enabling the disabled feature or fixing it by disabling any dependent features.

If the stakeholder adapting the UI is an internal employee, he or she will gain direct access to the adapted UI through the enterprise application. On the other hand UIs adapted by external experts remain in the online repository to be accessed by administrators from different enterprises. As the next subsections explain, due to business related usability and privacy matters, administrators are able to control the internally/externally adapted UIs that are made available to the enterprise employees and the internally adapted UIs that are shared with external communities.

Step 2: Administrator Checks, Integrates, and Publishes the Crowd-Adapted User Interface

The administrator of an enterprise application checks if the adapted UI matches the description given in the previous step. In case the UI had been adapted by one of the internal employees then the administrator would have access to it through *Cedar Studio* from the local database. Yet, if the UI was adapted by an external expert the administrator could download it using *Cedar Studio* from the online repository in XML format and import it to the local database.

Afterwards, the administrator associates the crowd-adapted UI with one or more enterprise roles (e.g., accountant, novice user, etc.). *Cedar Studio* automatically performs the role allocation to integrate the UI with our Role-Based User Interface Simplification mechanism (RBUIS).

Finally, the administrator publishes the crowd-adapted UI internally for the enterprise employees to use.

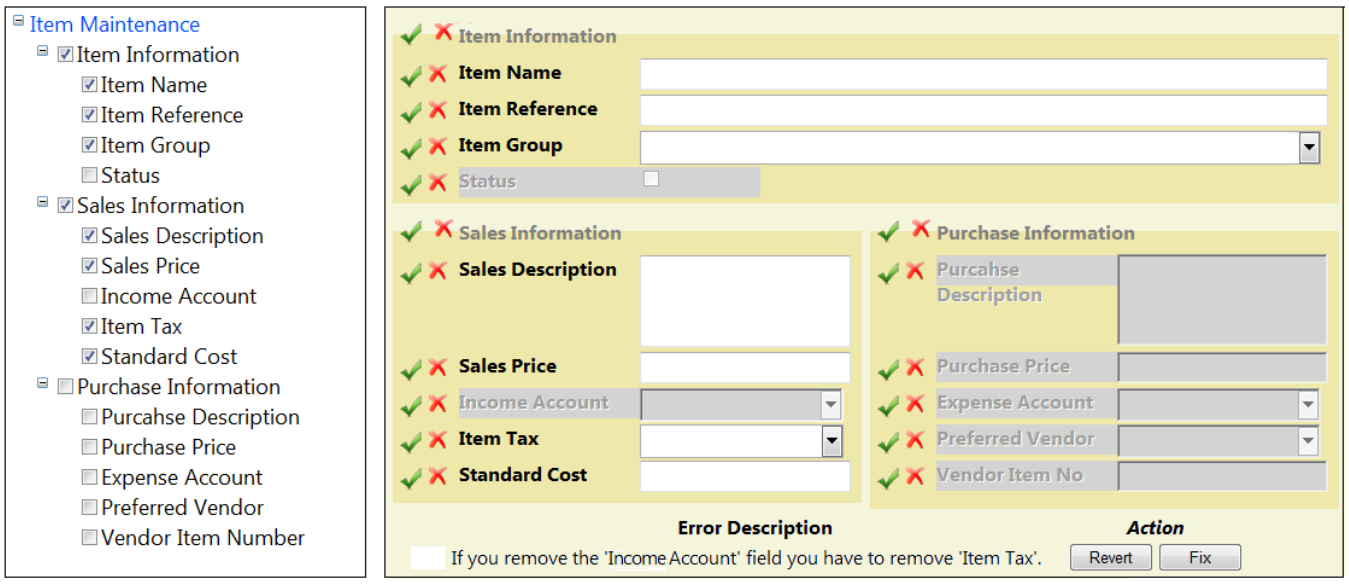

Figure 3. Our Web-Based Visual Feature-Set Editing Tool for Supporting Crowdsourcing User Interface Adaptations

Step 3: Enterprise Users Use and Rate the Adapted UI

After the administrator checks, integrates, and publishes the crowd-adapted UI, enterprise users with the appropriate roles will gain access to it. Although the crowd-adapted UI is ideally intended to provide a better user experience, the quality of the adaptation is always a concern. Hence, an end-user evaluation mechanism is needed to determine the adaptations that truly enhance usability for a given context.

After using the crowd-adapted user interface the users will be prompted to rate their user experience. One possible option to consider is the System Usability Scale (SUS) [6], which provides ten Likert-scale questions that could be converted into one numeric score.

Step 4: Administrators Share Internally-Adapted UIs and Internal Ratings

Due to privacy matters, some enterprises might decide not to share all the internally-adapted UIs and ratings. Hence, administrators are given control over which internally-adapted UIs and internally given ratings to share with the external communities.

In case the administrator decides to share an internally adapted UI, *Cedar Studio* could be used to upload the UI to an online repository alongside a description indicating the purpose of the adaptation. We should note that internally created enterprise roles are not shared with external communities due to their highly specific enterprise nature. Hence, the description fits as a substitution for these roles. Furthermore, ratings for internally or externally adapted UIs could be uploaded and aggregated with the rating data in the online repository to allow the external communities to benefit from this quality metric when searching for an adapted user interface that fits a particular context.

PRIVACY CONCERNS AND BUILDING COMMUNITIES AROUND ENTERPRISE APPLICATIONS

As we previously mentioned some enterprises might have privacy concerns regarding sharing some UIs, which have been internally adapted, with possible business competitors. Yet, this does not neglect the benefits of crowdsourcing UI adaptations. External experts could still contribute adapted UIs to online repositories for enterprises to benefit from.

Experts in commercial (e.g., SAP, Dynamics, etc.) as well as open-source (e.g., Compiere, A1, etc.) enterprise systems already contribute both knowledge and functionality to the enterprise communities. These experts contribute their knowledge to forums and gain a higher status (e.g., Microsoft MVP) in particular enterprise communities. Also, they contribute functionality by extending open-source applications and creating add-ons for commercial ones.

Enterprise applications already have numerous community networks (e.g., SAP Community Network [20]) where experts contribute their experience by helping other community members in solving enterprise application problems. Therefore, similar networks could be created for crowdsourcing the adaptation of enterprise application UIs. These networks could provide access to the feature-set editing tool (Figure 3), which could store the adapted user interface in the network's database thereby making the adaptations accessible online to any registered member.

Enterprises could also have an incentive for selling some proprietary adapted UIs on one of the enterprise application stores (e.g., Microsoft Dynamics Marketplace [21]). Some enterprises could even specialize in adapting and selling UIs for widely adopted enterprise systems. We should note that UIs developed with *Cedar Studio* could be easily shared in XML format due to their relational data nature.

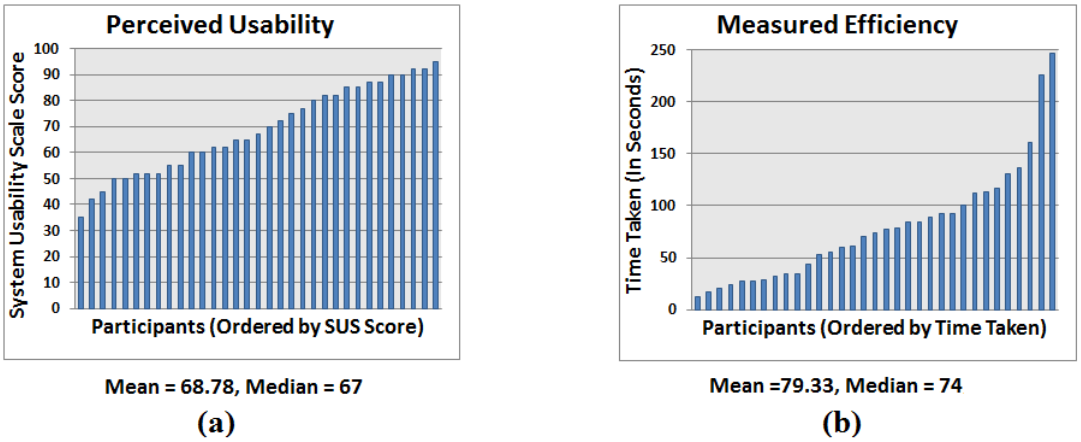

Figure 4. Results of the Study Conducted for Evaluating the Web-Based Visual Feature-Set Editing Tool

EVALUATION STUDY

In order to evaluate the approach that we are proposing in this paper, we made our feature-set editing tool available online and asked participants to adapt the feature-set of an "*Item Maintenance*" user interface as illustrated in Figure 3.

We used Amazon Mechanical Turk for crowdsourcing the adaptation task to 33 participants who were selected based on their Mechanical Turk experience and performance (>5000 hits and >95% accuracy). We diversified the sample by classifying participants into groups based on computer literacy. The participants were asked to rate their computer literacy through a series of questions based on an existing test [11]. The answers allowed us to classify participants as intermediate (13) and expert (20) computer users.

The participants were asked to minimize the feature-set based on given textual requirements describing the fields to be removed. After performing the adaptation, participants were asked to answer the System Usability Scale (SUS) questions to evaluate the usability of the tool. The task model resulting from the adaptation was stored alongside the time it took each participant to perform the adaptation. The stored information helps in assessing the efficiency and effectiveness of the participants when using the tool.

The results of the study are illustrated in Figure 4. Based on the given SUS scores (Figure 4 – a), with a mean score of 68.78, we can say that the participants perceived the system to be usable. Also, the participants were able to accomplish the given task successfully and efficiently with a mean time of 79.33 seconds (Figure 4 – b).

This basic preliminary study provides encouraging results in terms of the overall perceived usability and efficiency by participants with various computer skills. Yet, we should indicate that the study has some limitations in terms of the simplicity of the considered example and the selected participants. When the participants were asked if they would use such a tool in practice the majority agreed, nevertheless we are aware that Mechanical Turk participant could create some bias in terms of providing the researchers with the answers that they want to hear. Therefore, we are merely considering this study as a basic initial indicator and a pilot for future lab-based studies. In future studies we will consider more sophisticated examples from a specific enterprise application and we will recruit participants from the selected application's end-user community to test the tool. Based on the results of future studies we will be able say whether extending the tool can be worthwhile and possibly identify the new features that it should include.

CONCLUSIONS AND FUTURE WORK

In this paper we presented an approach for crowdsourcing UI adaptations by targeting the minimization of a UI's feature-set to reduce the "bloat" in enterprise applications.

Our approach relies on model-driven UI construction and making UIs available for the crowd to adapt through a web-based editing tool. To cater for privacy and quality concerns of enterprises, administrators are given a role in the adaptation process for controlling the externally adapted UIs that are published to the enterprise and the internally adapted ones that are shared with external communities. We argue that such concerns should not prevent online communities from forming around the proposed approach.

Our tool was evaluated through a preliminary online user-study that provided encouraging results in terms of perceived usability, and measured efficiency and effectiveness. Yet, we indicated the limitations in this study and our aim to overcome them in future lab-based studies.

In the future we could extend our web-based feature-set editing tool to support the adaptation of concrete UI widget properties (e.g., size, location, etc.). Also, we will test our tool with a real-life application by crowdsourcing UI adaptations to the application's relevant online community.

ACKNOWLEDGMENT

This work is partially funded by ERC Advanced Grant 291652.

REFERENCES

1. Akiki, P.A., Bandara, A.K., and Yu, Y. Using Interpreted Runtime Models for Devising Adaptive User Interfaces of Enterprise Applications. ICEIS'12, SciTePress (2012), 72-77.
2. Akiki, P.A., Bandara, A.K., and Yu, Y. RBUIS: Simplifying Enterprise Application User Interfaces through Engineering Role-Based Adaptive Behavior. EICS'13, ACM (2013), *Forthcoming*
3. Bergh, J., Sahni, D., and Coninx, K. Task Models for Safe Software Evolution and Adaptation. TAMODIA'09, Springer (2010), 72-77.
4. Bernstein, M.S., Brandt, J., Miller, R.C., and Karger, D.R. Crowds in Two Seconds: Enabling Realtime CrowdPowered Interfaces.UIST'11, ACM (2011), 33-42.
5. Botterweck, G. Multi Front-End Engineering. Model-Driven Development of Advanced User Interfaces. Springer (2011), 27-42.
6. Brooke, J. SUS: A Quick and Dirty Usability Scale. Usability Evaluation in Industry, Taylor and Francis (1996), 189-194.
7. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers, 15, 3, Elsevier (2003), 289-308.
8. Carroll, J.M. and Carrithers, C. Training Wheels in a User Interface. Communications of the ACM 27, 8, ACM (1984), 800-806.
9. Gajos, K. Automatically Generating Personalized Adaptive User Interfaces. Stanford University (2008). http://www.youtube.com/watch?v=ODrE7SodLPs.
10. Heimerl, K., Gawalt, B., Chen, K., Parikh, T.S., and Hartmann, B. CommunitySourcing: Engaging Local Crowds to Perform Expert Work via Physical Kiosks. CHI'12, ACM (2012), 1539-1548.
11. Kay, R.H. A Practical Research Tool for Assessing Ability to Use Computers: The Computer Ability Survey (CAS). JRCE 26, 1, IACE (1993), 16-27.
12. Lafreniere, B., Bunt, A., Lount, M., Krynicki, F., and Terry, M.A. AdaptableGIMP: Designing a Socially-Adaptable Interface. UIST'11, ACM (2011), 89-90.
13. McGrenere, J., Baecker, R.M., and Booth, K.S. An Evaluation of a Multiple Interface Design Solution for Bloated Software. CHI'02, ACM (2002), 164-170.
14. McGrenere, J. and Moore, G. Are We All In the Same "Bloat"? Graphics Interface, A.K. Peters (2000), 187-196.
15. McGrenere, J. "Bloat": The Objective and Subject Dimensions. CHI'00, ACM (2000), 337-338.
16. Nebeling, M., Leone, S., and Norrie, M. Crowdsourced Web Engineering and Design. Web Engineering. Springer (2012), 31-45.
17. Nebeling, M., Leone, S., and Norrie, M. Crowdsourced Web Engineering and Design. ICWE'12, Springer (2012), 31-45.
18. Paternò, F., Mancini, C., and Meniconi, S. Concur TaskTrees: A Diagrammatic Notation for Specifying Task Models. INTERACT'97, Chapman & Hall (1997), 362-369.
19. Shneiderman, B. Promoting Universal Usability with Multilayer Interface Design. CUU'03, ACM (2003), 1-8.
20. SAP Community Network. http://scn.sap.com/welcome.
21. Microsoft Dynamics Marketplace. http://dynamics.pinpoint.microsoft.com/en-GB/home.

APPENDIX

Algorithm 1. Conflict Checking for Feature-Set Minimization Based on CTT Temporal Constraints

```
// m = number of unselected tasks, n = number of conflicting tasks
// CON = Constant, POL = Polynomial, c1 … c9 = cost1 … cost9
                  [] CheckForConflicts(TaskModel TM) // Running Time = O(m)
                  {//Get the unselected tasks and their relevant relationships
CON   c1    O(1)      UnselectedTasks[] ← Select * From TM.Tasks Where Selected == false
CON   c2    O(1)      UnselTaskRelationships[] ← Select * From TM.Relationships as R
.     c2    O(1)      Where (Select TaskID From UnselectedTasks).Contains(R.SourceTaskID)
.     c2    O(1)      || (Select TaskID From UnselectedTasks).Contains(R.TargetTaskID)
.     .     .         //CTT Rel. types that indicate dependency between tasks (TA & TB)
CON   c3    O(1)      RemoveTAIfTBIsRemoved[] ← { Concurrency with Info. Exchange }
.     c3    O(1)      RemoveTBIfTAIsRemoved[] ← { Concurrency with Info. Exchange,
.     c3    O(1)                                  Enabling, Enabling with Info. Exchange }
CON   c4    O(1)      ConflictingTasks ← [];
POL   c5    O(m)      foreach uTask in UnselectedTasks
.     .     .                 //Get the conflicts created by unselecting the task
CON   c6    O(1)              ConflictingTasks ← Select * From TM.Tasks as T Where
.     c6    O(1)              (Select SourceTaskID From UnselTaskRelationships
.     c6    O(1)              Where TargetTaskID == uTask.TaskID
.     c6    O(1)              && RemoveTAIfTBIsRemoved.Contains(RelType)).Contains(T.TaskID)
.     c6    O(1)             ||(Select TargetTaskID From UnselTaskRelationships
.     c6    O(1)              Where SourceTaskID == uTask.TaskID
.     c6    O(1)              && RemoveTBIfTAIsRemoved.Contains(RelType)).Contains(T.TaskID)
CON   c7    O(1)      return ConflictingTasks
```